SIMULATION-BASED FUNCTIONAL VERIFICATION OF MICROCIRCUIT DESIGNS

COPYRIGHT NOTICE

BACKGROUND OF THE INVENTION

The invention relates to systems and methods for verifying the design of digital microcircuits. Such digital microcircuits are now commonplace, and are used in a wide variety of applications, from desktop computers to television controls and common appliances to controllers for machinery, sophisticated weapons systems, and supercomputers. The digital microcircuit design to be tested shall be referred to as the design under test (DUT).

More particularly, the present invention relates to systems and methods applied to a high-level DUT model with the purpose of generating test data that, when applied to a simulation of the DUT, achieves high levels of DUT coverage and therefore has a high likelihood of finding any errors in the DUT.

One known approach has been the use of random input generators to provide input sequences to randomly "drive" the DUT model through the "state space" of possible input and internal register combinations, in the hope of feeding the model a sufficient variety of input combinations to make it possible to identify any errors in the design of DUT. Such methods provide for relatively deep penetration of the space of possible states, but do not provide acceptably broad coverage. This difficulty is aggravated by the observed tendency of even sophisticated random input generators to provide identical or very similar input sequences, or otherwise produce identical or similar DUT register states, on a repeated basis.

Other attempts have involved the use of exhaustive formal search methods. Such methods provide potentially complete coverage of the state space, but for even moderately complex DUTs the state space is so large that time and resource limits preclude the exclusive use of such methods from being effective.

The effect of prior art analysis methods are shown schematically in Figure 1a. State space 20, which represents in two-dimensional form the space of possible input and register states of the DUT, comprises a plurality of goal states 11 and a start state 10, which typically represents the reset state for the DUT. Trace 15 represents the path through the state space followed by the DUT model in being driven by randomly-generated inputs. Trace 15 wanders relatively deeply through the state space, but without breadth and without apparent direction, and loops back over itself or near to itself at points 23. Trace 15 has intersected two goal states 13 and missed many others. Step traces 31 represent state sequences simulated in the DUT by means of formal methods. Traces 31 have covered a relatively broad but shallow region of the state space, and have located both goal states 13 located within the coverage region.

In speaking of circuit verification by computer simulation, it is useful to define several terms. A goal or coverage state is any state which is desired to be reached under test conditions to provide adequate coverage of the DUT. Typically there exist a number of tasks a circuit is designed to do, and a number of things that are recognized as undesirable for the circuit to do – that is, there exist a number of circuit states which it is desirable or undesirable for the circuit to assume. Usually a set or record of such desirable and undesirable states is maintained and used as a basis for coverage measurement and for analysis of the quality of testing. While most generally a goal state completely defines the complete register contents of the DUT, a goal state definition may specify only a subset of the register bits, and therefore specifies a boolean hyper-cube of specific states.

A formal simulation method is any method used to systematically – as opposed to randomly – simulate different states of a DUT. In particular, formal simulation methods include formal reachability methods. Formal reachability methods, in the context of this application, include any exhaustive or other mathematical techniques for finding input assignments for causing a state machine, or a state machine model, to transition from a starting state to some other state. Examples of formal simulation methods include symbolic simulation and SAT bounded model checking techniques, as disclosed herein.

BRIEF SUMMARY OF THE INVENTION

The invention provides system, methods, and apparatus for verifying microcircuit designs by interleaving between random and formal simulation techniques to identify input traces useful for driving DUT models through a set of goal states.

In one aspect the invention provides a method of determining an input sequence for verifying a design for a microcircuit. The method comprises beginning random simulation of a sequence of states of a microcircuit design by inputting a sequence of random input vectors to a DUT model in order to obtain a sequence of random simulation states; monitoring a simulation coverage progress metric to determine a preference for switching from random simulation to formal methods of simulating states in the DUT; beginning formal simulation of states in the DUT model and monitoring a formal coverage progress metric to determine a preference for resuming random simulation of states of said microcircuit design; and resuming random simulation.

Preferably the DUT model used by the formal and random simulation methods are the same, or at least highly compatible, in order to save modeling and computation time.

In preferred methods according to the invention the process of switching from random to formal analysis methods and back again, or "interleaving", is continued until inputs usable for driving the DUT model into each of a set of previously-defined goal states have been identified.

Preferred formal methods for use with the invention comprise symbolic simulation and satisfiability techniques. These methods are used both singly and in combination in practicing the invention. One of the particular improvements offered by the invention is to start formal analysis from a DUT model state identified by random simulation, instead of a reset or other arbitrary state. This promotes "deep" coverage of the state space.

The interleaving process is one of the most advantageous aspects of the invention. By switching back and forth between formal and random simulation methods, and particularly from states identified through random simulation, both broad and deep coverage of the DUT state space may be obtained and an improved confidence in coverage of the DUT gained.

The effect of the interleaving process of the invention in improving coverage of the DUT state space is illustrated schematically in Figures 1b and 1c. State space 20 represents in two-dimensional form the space of possible input and register states of the DUT, and comprises a plurality of goal states 11 and a start state 10 which typically represents the reset state for the DUT. Trace 15 represents the path through the state space followed by the DUT model in being driven by randomly-generated inputs. Trace 15 wanders relatively deeply from initial state 10 through the state space, but without breadth and without apparent direction, and loops back over itself or near to itself at points 23. In the case shown in Figure 1b trace 15 has intersected two goal states 13, but after striking goal state 17 has failed to locate any others. After simulating from goal state 17 to point or state 19 random simulation is stopped and formal simulation begun from last-reached state 19. Steps 31 represent state sequences simulated in the DUT by means of formal simulation. Eventually the formal method intersects goal state 18 and random simulation is resumed. Second random trace 15' identifies additional goal states 22 before proceeding fruitlessly to trace end 19', at which point formal simulation is resumed, eventually to identify goal state 18'. The interleaving process is preferably repeated, and provides both breadth and depth in searching state space 20 for goal states. An alternative interleaving process is shown in Figure 1c. In Figure 1c random simulation is resumed from last-reached goal states 17 and 17' instead of from ends 19 and 19' of traces 15 and 15'.

Additional aspects of the invention comprise optional starting (or re-starting) of the random simulation process, during the interleaving process, through the use of a trace generated by formal simulation; and starting of formal simulation from a state generated by random simulation.

Optionally the invention further comprises using formal methods to prove one or more goals states of the DUT to be unreachable, in order to maximize efficiency and avoid the use of CPU and memory resources in searching for input vectors to states which cannot be reached. Generally this is accomplished by passing information generated by the unreachability engines to formal reachability engines.

In other aspects the invention provides such methods as performed by data processing systems, data processing systems for performing such methods, and computer program products comprising computer usable media having computer readable code embodied therein for causing computers to perform such methods.

It is important to note that except insofar as a particular order of steps of any method or process described herein is inherent, or it is otherwise stated expressly that any given combination of steps must be completed in a given order, no order to the steps of any method or process described herein is implied or required.

BRIEF DESCRIPTION OF THE DRAWINGS

The invention is illustrated in the figures of the accompanying drawings which are meant to be exemplary and not limiting.

Figure 1a is a schematic diagram of prior art simulation techniques. Figure 1b is a schematic diagram of a preferred interleaved simulation techniques according to the invention. Figure 1c is a schematic diagram of alternative preferred interleaved simulation techniques according to the invention.

Figure 2 is a schematic representation of a preferred process for simulation-based functional verification of microcircuit designs according to the invention.

Figure 3 is schematic diagram of a program structure for implementing on a computer system a simulation-based functional verification process according to the invention.

Figure 4 is a schematic representation of control flow of a preferred process according to the invention as executed by a suitable data processing system.

Figure 5 is a pseudocode representation of a portion of a preferred embodiment of a computer-implemented random simulation process according to the invention.

Figure 6 is a schematic representation of a timing cycle in a simulation process according to the invention.

Figure 7 is a pseudocode representation of a process for controlling interleaving between analysis methods according to the invention.

BRIEF DESCRIPTION OF THE PRINTED APPENDICES

The accompanying printed appendices, which are incorporated as a part of this specification, illustrate several aspects and embodiments of the invention, and together with the description serve to explain the principles of the invention.

Appendix 1 (50 pp.) is entitled "Design and Maintenance Specification for CTG Reachability & Control Subsystems" and describes a preferred embodiment of a data processing system adapted to perform verification simulation processes according to the invention.

Appendix 2 (427 pp.) is entitled "Vera™ Verification System User's Manual

Version 4.0" and describes the Vera™ verification system.

Appendix 3 (7 pp.) is entitled "Smart Simulation Using Collaborative Formal and Simulation Engines," by Pei-Hsin Ho et al. and describes a method of performing unreachability analyses.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

Reference will now be made in detail to preferred embodiments of the invention, examples of which are illustrated in the accompanying drawings. Wherever possible, the same reference numbers will be used throughout the drawings to refer to the same or like parts.

Figure 2 illustrates a preferred process 100 for verification of a microcircuit design according to the invention. At 102 a model of the circuit (the DUT model) is defined, and at 104 an initial state is applied to the model so that the circuit is "driven" into a model of the design in a selected initial state. It is useful in many instances to model the inputs for this initial state so that their application to the DUT model results in the model being driven into a design reset state for the circuit, so that the model is completely defined in a valid, useful starting state. However, many other suitable starting states are also possible and suitable, depending upon the particular analysis tools to be used. A set of one or more goal states for the design is defined at 106. It is also useful in many circumstances, such as will be plain to those of ordinary skill in the art, that in many cases the use of separate but parallel DUT models for one or more of the various formal search processes will be preferable and more efficient.

Depending upon the requirements of a particular analysis or test situation, it may be advantageous to generate either a single DUT model for use in all simulation processes, or a plurality of such models, for example one for each distinct random and formal simulation method. The factors involved in deciding how many DUT models to make, and of what type, are matters well within the ability of the designer or analyst skilled in the art.

At 108 a process of checking to see whether all goal states have been reached (or, in cases in which unreachability analyses are used, have been either reached or identified as unreachable) is started. On the initial pass no goal states have either been reached or proved not reachable, and random analysis is started. An initial step in this process is the generation at 110 of a random input sequence for the design model. In many cases this input sequence takes the form of a vector.

At 112 the random input sequence is mapped to a form which can be considered valid within the operating environment expected to be experienced by the DUT. In general, when a circuit is designed it is anticipated that the circuit will operate in an environment in which certain things can be expected to happen, while certain other things will be expected not to happen. For example, a microprocessor might be expected to comply with a known bus protocol, or to receive or to not receive certain types of input from top level programs, etc. It is generally advantageous to acknowledge this and increase the efficiency of the analysis by ensuring that only valid inputs are applied to the design model.

At 114 the input sequence generated at 110 is applied to the simulated circuit model and the state of the model under such input is determined. This is referred to as "driving" the circuit model. At 115 the input sequence used to drive the model is recorded in the vector trace set. This input set or vector trace generally comprises all inputs required to reproduce the goal state simulation in the DUT model, and ultimately all goal states reached by the simulation verification process. In many cases is it convenient and efficient to record each state into which the circuit model is driven, regardless of whether the state is a goal state or not. This is one certain way of ensuring that each state reached by the circuit model during simulation may be recreated during later DUT testing. In other circumstances it may be advantageous to record input traces only when goal states have been reached.

At 116 the state result obtained by driving the circuit model with the random input sequence is checked against the set of goal states defined at 106. If the state is a goal state, then the fact that that state has been reached is recorded and the record of goal states is checked at 108 to determine whether all goal states have been reached. If all goal states have been reached, the trace capable of driving the circuit model into each of the goal states is ready for further use, most typically for verification of the DUT. If all goal states have not been reached, then the random simulation process resumes at 110.

If the state reached by driving the design model at 114 with the random input sequence generated at 110 is not a goal state, a simulation coverage progress metric is evaluated at 118 to determine whether sufficient progress towards identifying coverage states is being made. If the simulation coverage progress metric is not satisfied, random simulation is stopped and formal analysis is initiated. If the simulation coverage progress metric has been satisfied, then random simulation is resumed from 110.

A great many simulation coverage progress metrics are suitable for use with the processes disclosed herein. A simple but highly effective class of simulation coverage progress metrics comprises simple monitoring of loop counters. For example, one preferred method for determining when to switch to formal methods is to count the number of times the random simulation process has been applied without identifying inputs for a goal state, and to switch to formal simulation methods when the process has failed a certain number of times. This is readily accomplished by setting a counter to an initial value on starting the random analysis process, resetting the counter each time a goal state is reached, and incrementing or decrementing the counter each time the path from 110 and 118 is covered without identifying a goal state. When a predetermined number of random simulation attempts to drive the model into a goal state have failed, random analysis is stopped and formal analysis is initiated. This serves to prevent the inefficient use of resources such as CPU time and memory.

A second highly useful class of simulation coverage progress metrics includes those based on the relative distinctiveness, or "freshness," of the state or set of states most recently induced in the DUT model. For example, one of the most useful variations in this class is that in which a percentage of times that the random input process has induced a given complete or partial state in the circuit model is monitored, and in which when (or if) such percentage exceeds a predetermined value random simulation is halted and analysis is shifted to formal methods. This can be particularly useful given the observed tendency of many random input generation algorithms to provide repetitive or substantially repetitive output over a series of generation cycles. One of the most beneficial manners in which to implement this type of simulation coverage progress metric is to monitor only selected state variables within the circuit model (that is, a "cube", or group of states).

It is also beneficial in some circumstances to avoid the rote application of simulation coverage progress metrics, by temporarily suppressing or ignoring them, or by using them in combination to create hybrid metrics. For example, it is sometimes useful to suppress loop counters or other counting means for determining coverage progress and to allow the random process to continue for additional time if a relatively "fresh" state has been induced in the circuit model. This helps to assure a suitable minimum level of exploration once fresh states have been reached.

When a simulation coverage progress metric has not been satisfied, random simulation is stopped and formal methods are initiated. Typically the last state reached by simulation is used as a start state for formal methods, although other states (such as the last goal state reached) may be used. Thus the process continues at 120 with selection of a start state for use in initiating formal analysis. This start state is made available for use in formal analysis.

Many formal analysis methods suitable for use with the invention are known. Two formal analysis methods that are particularly suitable for use with the invention are those known as (a) symbolic simulation and (b) satisfiability ("SAT") techniques, particularly SAT techniques used in conjunction with bounded model checking (BMC) techniques. As used separately, both symbolic simulation and SAT techniques are well known. Each has its own advantages and disadvantages. Preferably both methods are used, so that their strengths can be exploited to the greatest possible degree.

Symbolic simulation techniques suitable for use with the processes disclosed herein are described in "Symbolic-Simulation -- Techniques and Applications," by R.E. Bryant, in Proceedings of DAC, pp. 517 - 521, 1990; "Test Pattern Generation for Sequential MOS Circuits by Symbolic Fault Simulation", by K. Cho and R.E. Bryant, in Proceeings of DAC, pp. 418-423, 1989; and "Cycle-based Symbolic Simulation of Gate-Level Synchronous Circuits" by V. Bertacco, M. Damiani and S. Quer, in Proceedings of DAC, pp 391 - 396, 1999, each of which is herein incorporated by reference. SAT techniques suitable for use with the processes disclosed herein are described in "Symbolic

Model Checking Using SAT Proceedures" by A. Biere, A. Cimatti, E. Clarke, M. Fujita, and Y. Zhu, in Proceedings of DAC, 1999; and "Verifying Safety Properties of a PowerPC Microprocessor Using Symbolic Model Checking without BDDs", by A. Biere, E. Clarke, R. Raimi, and Y. Zhu, in Proceedings of CAV, 1999, each of which is herein incorporated by reference.

Figure 2 illustrates the case in which symbolic simulation is selected as the initial formal method. Optionally, satisfiability or other techniques may be used as the initial formal method. Symbolic simulation is initiated at 122. Symbolic simulation drives each primary input of the design model with a new symbolic variable at each simulation step, each simulation step corresponding to a selected number of modeled clock cycles, the number of cycles selected being dependent upon the design being modeled and the manner in which it has been modeled. A symbolic formula is generated for each signal or state variable, according to the logic in the signal's or state variable's fanin. Signal and state variable values are recorded as Binary Decision Diagrams (BDDs). Given a starting state, as selected at 120, symbolic simulation steps through simulated time to reach in "i" number of steps any new coverage states within "i" steps of the starting state. After each step the state variables are checked at 124 to determine whether a new goal state has been reached by substituting the coverage signals in a BDD representation of an unreached goal state with the coverage signals generated by the symbolic simulation process. If the state is a goal state, then the fact that that state has been reached is recorded and the set of inputs for the model required to produce the goal state result (i.e., the "trace" which produces that result) is used at 140 to drive the design model into the goal state. This input set is also preferably recorded in the vector trace set. The record of goal states is then checked at 108 to determine whether all goal states have been reached. If they have, a set of traces capable of driving the circuit model into each of the goal states is recorded for further use, most typically as the design under test is refined. If all goal states have not been reached, the random simulation process is resumed at 110.

If the state reached by driving the design model at 122 with symbolic simulation is not a goal state, the current status of the analysis is evaluated at 126 to determine whether any one of a set of one or more formal coverage progress metrics has not been satisfied. If the formal coverage progress metric has not been satisfied, symbolic simulation is stopped and SAT bounded model checking is initiated at 128. If the formal coverage progress metric has been satisfied, the symbolic simulation process is repeated from 122.

Preferred formal coverage progress metrics for use with symbolic simulation include limits selected by the circuit designer on CPU or other calculation time and/or available memory or recording resources. Again, the major disadvantage to using systematic formal methods to test the entire design is that the evaluation of even a modest modern design requires the use of large amounts of time and resources, and therefore makes the sole use of such methods expensive, and generally inconvenient. For most designers time and analysis resources such as computer time and memory are limited. Typically, then, formal methods are used only to locate, if possible, a new coverage state from which to resume random analysis, within explicit CPU and memory limit constraints.

SAT analysis is initiated at 128. Given a particular starting state, SAT is used to search for new goal states within a given number of analysis steps. SAT uses Boolean representations of portions or preferably all of the circuit model, each representation being modeled as a function f(x), and searches for assignments for state variables x, f(a), such that f(a) = 1, f(a) = 1 being defined so as to correspond to a goal state. When a goal state has been reached, then the fact that that state has been reached is recorded and the set of inputs for the model required to produce the goal state result (i.e., the "trace" which produces that result) is used at 140 to drive the design model into the goal state. This input set is also preferably recorded in the vector trace set. The record of goal states is then checked at 108 to determine whether all goal states have been reached. If they have, a set of traces capable of driving the circuit model into each of the goal states is recorded for further use, most typically as the design under test is refined. If all goal states have not been reached, the random simulation process is resumed at 110.

If the state reached by driving the design model at 128 through a single step of SAT simulation is not a goal state, the current status of the analysis is evaluated at 132 to determine whether any one of a set of one or more formal coverage progress metrics has been satisfied. If any one or more of the formal coverage progress metrics has not been satisfied, SAT simulation is stopped and random simulation is resumed at 110. If the formal coverage progress metric has been satisfied, the SAT simulation process is repeated from 128.

Preferred formal coverage progress metrics for use with SAT analysis include limits selected by the circuit designer on CPU or other calculation time and/or available memory or recording resources. Such limits are suggested by, for example, test schedules and the nature of available computational resources. Again, the major disadvantage to using SAT methods to test the entire design is that the evaluation of even a modest modern design requires the use of large amounts of time and resources, and therefore makes the use of such methods expensive and generally inconvenient. For most designers, time and analysis resources such as computer time and memory are limited. Typically, then, they are used only to locate a new coverage state from which to resume random analysis, within explicit CPU and memory limit constraints.

In general random simulation may be seen as an effective tool for simulating through long traces, that is, through relatively large numbers of time steps; but is weak at developing or investigating the existence of possible states within any definite specified numbers of time steps, at which formal methods excel. Of the formal methods, symbolic simulation is most effective with designs having fewer input variables, and is capable of searching effectively for goal states over medium-length traces. SAT methods are most effective at exhaustively searching through short traces.

Process 100 proceeds through the steps described, interleaving between random and formal simulation techniques and back again, until it is determined at 108 that each of the goal states defined at 106 has been reached, at which point a set of input traces suitable for driving the design model into each of the goal states has been recorded. This trace record is used in further evaluating the DUT, particularly where design revisions are ongoing and it is desired to ensure that subsequently revised designs function as intended.

Figure 3 is a schematic diagram of a program structure for implementing on a computer system a verification simulation process according to the invention, as for example the process shown schematically in Figure 2. System 150 comprises user interface 152, master control process 154, and slave processes 156, which comprise random simulation module 158, symbolic simulation module 160, SAT BMC module 162, and unreachability module 164. User interface 152, master control process 154, and slave processes 156 comprise either separate, stand-alone programs or modules forming a single program, depending upon the needs and convenience of the circuit designer and analyst for the particular problem at hand. A particular advantage offered by implementing the user interface, master process, and slave processes as separate programs or program modules for the various analysis components is that the concurrent operation of separate program functions is facilitated. For example, as described herein it is often advantageous to provide an unreachability module to quickly determine a large number of unreachable states, the identities of which may be used to increase the efficiency of the random and formal simulation modules. It has been found to be advantageous to execute the unreachability module simultaneously with the simulation modules. This can be accomplished, for example, through time-sharing or parallel processing techniques.

User interface 152 is provided in any one of a wide variety of forms, but is typically a general purpose operating system, adapted for interactive use (although batch processing is also useful for some applications). In general, user interface 152 facilitates employment by the user of whatever decision making and control processes are required for completion of the analysis at hand, and is responsible for presentation to the user of data received from or through the master control process. Input from and output to the interface can be graphics or text-based, depending upon the needs and convenience of the user. An example of an operating system suitable for adaptation for use with computer-implemented processes according to the invention is UNIX.

Master control process 154 coordinates and controls the operations of slave processes which perform the bulk of the verification simulation. In addition, heuristics adapted to assist the user in selecting various random and formal simulation processes during the analysis are incorporated within or made available to the master control process. Preferably the master process facilitates the concurrent or sequential operation of the slave processes, so as to maximize the efficiency of the simulation effort. Preferred master processes also continually monitor input and output messages and commands from slave and user interface modules.

Slave processes 156 represent various reachability and optionally unreachability modules (sometimes called engines), each preferably run as a separate process. Each of the slave processes uses specialized algorithms to search for sequences of inputs which drive the DUT from one state to another in a search for goal states (or, in the case of unreachability engines, to prove various goal states unreachable), and each is adapted to receive input commands from the master process and to report progress and various control flag conditions to the master control process. A suitable means for transfer of input commands and for reporting of output to and from the slave processes is through an operating system such as UNIX. Slave processes 156 comprise random simulation module 158, symbolic simulation module 160, SAT simulation module 162, and unreachability module 164.

Random simulation module 158 comprises a biased, constrained, random input generator adapted to drive the circuit model through its state space in an attempt to identify input patterns which reach goal states. It has the ability to generate start states for searches performed by formal search modules, to apply to the DUT model sequences of inputs which are generated by the formal engines, and to save the entire vector trace set for later use in validating the DUT. A number of random simulation programs suitable for use as random simulation module 158 are available commercially, for example the VCS Verilog simulator, available commercially from Synopsys, Inc., of Mountain View, California.

It is characteristic of the invention and of the embodiments thereof described herein that they are well suited for integration with and use of a number of pre-programmed analysis programs and modules available commercially from a number of companies and sources. For example, ModelSim of Mentor Graphics, Verilog-XL and NC Verilog of Cadence, and Scirocco of Synopsis are well suited for use in the analysis processes and systems decribed herein.

Symbolic simulation module 160 uses symbolic techniques as discussed herein to attempt to reach goals from a given starting state. If a goal is identified, the engine saves a sequence of inputs which are replayed by the random simulation module to prove that the state has been reached.

SAT module 162 uses bounded model checking concepts, together with classic satisfiability techniques, to reach goals from any given starting state. If successful, the engine saves a sequence of inputs which are replayed by the random simulation module to prove that the state has been reached.

Unreachability module 164 uses any of various formal techniques to prove that certain goal states are not reachable from a given start state. The start state which is most often of greatest interest is the reset state of the DUT. States shown to be unreachable are then optionally communicated to the random and formal simulation modules to reduce the size and complexity of the search problem. A preferred method for unreachability analysis employs symbolic fixed point computation techniques, in which the unreachability engine computes an approximate fixed point for reachable states, then inverts the fixed point as a proof of unreachable states. The fixed point computation is typically overreached. Stated another way, fixed point symbolic image computation computes the set of states that can be reached in one clock cycle from a given starting state, iterating systematically through clock cycle steps until no new states are added. Analytic techniques suitable for use in unreachability analyses according to this aspect of the invention are described in "Formal Verification of Pipeline Control Using Control Token Nets and Abstraction Interpretation," by Pei-Hsin Ho, Adrian J. Isles, and Timothy Kam, in ICCAD 1998, pp. 529-536, ICCAD, 1998. This document is herein incorporated in full, by this reference. An alternative method for unreachability analysis, in addition to that shown in the foregoing incorporated reference, is described in Appendix 3.

After the unreachability analysis is completed the master control process is provided a set of states proved unreachable and these states are removed from the list of outstanding goal states sought by reachability engines.

Figure 4 illustrates a representative control flow of a preferred process according to the invention as executed by a suitable digital computer system, such as the system depicted schematically in Figure 3. To clarify discussion, process 200 is illustrated without showing user interaction processes such as those which might be received via user interface 152 in Figure 3 to structure or control the simulation process.

The process illustrated in Figure 4 is generally similar to that illustrated in Figure 1, except that an optional unreachability algorithm is used to improve the efficiency of the process and to broaden coverage of the DUT state space by proving a large number of potential states unreachable. This information, passed to and used by the formal analysis module, increases the efficiency of the formal search.

Process 200 of Figure 4 begins at 201 with invocation of the master control process by a user, via the user interface. The master control process first loads the DUT model and a set of goal states.

At 202 the master control process provides instructions to initialize the symbolic simulation module. At 203 the symbolic simulation module reports readiness to the master control program and stops pending receipt of further instructions.

At 204 the master process starts the random simulation module. The random simulation engine is initialized and loads goal states, performs a reset sequence to drive the DUT model into the reset state, and then performs a number of random input simulation cycles. As described above the random analysis simulator monitors the reaching of goal states until at 205 a simulation coverage progress metric is no longer satisfied and control is shifted back to the master control process. The current DUT model state, the last goal state reached, or another suitable starting state for initiation of symbolic simulation is written as output by the random simulator for input to the symbolic simulation engine.

At 206 the master control process, on the basis of various heuristics and optionally under interactive control via the user interface, decides how to continue the verification process. It having been observed that it is often beneficial to begin verification with a number of random searches from the reset state, it is decided, at 207 in the example shown, to return to random simulation, and the master control process issues suitable commands.

At 208 the random simulation process resumes. As before, random simulation continues until, at 209, a simulation coverage progress metric is no longer satisfied and control is shifted back to the master control process. Again the current DUT model state, the last goal state reached, or another suitable starting state for symbolic simulation is written as output by the random simulator for input to a formal methods engine.

At 210 the master control process determines heuristically to commence use of formal methods instead of returning to random simulation, and selects in particular symbolic simulation. At 211 random simulation module 158 is instructed to drive the DUT into the reset state; upon reporting of completion of the reset function by the random simulation module at 212 the symbolic simulation module 160 and the unreachability module 164 are instructed by the master control process to load the start state saved at 211 and begin analysis.

At 213 the unreachability module starts. The unreachability module loads the goal state set, then performs a fixed point computation from the start state provided by the random simulation module, in this example the reset state, which is in many cases the most efficient state from which to begin. After the fixed point for the beginning of analysis has been computed, those goals outside the fixed point are proved unreachable. As unreachable states are identified they are written to disk for later use in saving wasted effort by other slave modules. Preferably the unreachability module runs in parallel (as for example by means of parallel processing or time sharing techniques) with the random simulation and formal methods or other reachability processes, in order to provide unreachability data as early as possible and thereby to minimize wasted effort by the other analysis modules.

At 214, concurrently with starting of the unreachability module, a formal search is initiated using the symbolic simulation engine, beginning from the start state identified at 211, in this example again the reset state. At 215 the symbolic simulation module finds a goal state and saves a trace thereto, for use by the random simulation module in driving the DUT model to the state identified, and control is returned to the master control process at 216.

At 216 the master control process instructs the random simulation module to play the trace generated by the symbolic simulation module at 215 and to resume random simulation from that state. At 217 the random simulation engine replays the trace and resumes random searching for and recording of goal states.

At 218, while the random simulation module is searching for and recording identified goal states, the unreachability module completes its analysis and writes to a memory (readable by the other slave processes) a set of unreachable states for future use in maximizing the efficiency of formal methods. At 219 - 220 the results of the unreachability analysis are used to adjust the list of coverage goals and the global simulation coverage progress metric used to determine when all reachable goal states have been identified, to prevent searching for unreachable goals. When the list has been adjusted, the master control process stops pending further results from the random simulation module. The relative location on the chart of points 218 – 220, at which the unreachability process ends and results are reported to the master process, for use by the reachability engines, is not fixed. One or more cycles of random simulation and / or either or both formal engines may occur before unreachability results are completed and reported.

At 221 a simulation coverage progress metric is no longer satisfied by the random simulation module and control is returned to the master control process. The current DUT model state, last goal state reached, or other suitable starting state for symbolic simulation is written as output by the random simulator for input to the symbolic simulation engine.

At 222 the master control process instructs the symbolic simulation module to resume, from the state provided at 221 by the random simulation module. Symbolic simulation continues unsuccessfully – that is, without identifying any goal states – until a coverage progress metric is no longer satisfied, and at 223 the master control process is notified.

At 224 the master control process heuristically determines to continue formal searching through the SAT module, and instructs the SAT module to begin from the state provided at 221 by the random simulation module. The SAT simulation process continues until at 225 a goal is reached. A trace for driving the DUT model to the newly-identified goal state is recorded and control is returned to the master process, which at 226 restarts the random simulator, instructing it to replay the trace and resume.

The process of interleaving formal and random simulation methods, for example as depicted between 221 and 226, continues with another random simulation cycle stopping at 227, another symbolic simulation cycle at 228 – 229, and a repetition of the cycle at 230 - 231 until at 232 a determination is reached that all goal states have either been reached or are not reachable, upon which a set of traces for driving the DUT to each reachable state is written and at 233 control is returned to the master control process, which informs the user and stops the analysis.

It is important to note that the particular order of random and formal simulation techniques used in the interleaving process is highly flexible and dependent upon the design or analysis task at hand. The interleaving order described above, while useful for many circuit analysis applications, and often times preferred is for purposes of this disclosure exemplary only.

EXAMPLE

A data processing system is configured to perform a verification simulation process according to the above. The system comprises distinct program modules as shown in Figure 3, including a user interface 152; a master control process 154; and random simulation, symbolic simulation, satisfiability, and unreachability slave processes 158, 160, 162, and 164 respectively. The system and its operation are described in the Design and Maintenance Specification for CTG Reachability & Control Subsystems attached hereto as Appendix 1.

The user interface is interactive and comprises the UNIX operating system, including standard UNIX commands such as “fork’ and ‘exec’. Commands are issued to the master control process via a UNIX pipe which feeds inputs to the master control process as standard input. The user interface may operate the system in either graphical or textual mode.

Master control process 154 is written in the C programming language, and uses a Task Control Language (“TCL”) command interpreter for accepting commands from the user interface process. The master control process is design independent, so that a single master control process may be used in the analysis or verification of a number of different design models.

The Programming Language Interface (“PLI”), which comprises a set of standard techniques and function calls provided by IEEE standard Verilog simulators, is used for several purposes, including the monitoring of goal coverage status; allowing access by the master control process to the state of the DUT model, for downloading to the formal simulation modules; reporting covered goal states; determining and monitoring simulation progress coverage metrics; and tracking the freshness of DUT model states.

Flow control for the simulation process is embodied in TCL procedures. TCL is interpreted by the hybrid verification (sometimes herein "hv") program, which is run in all slave processes (except the random simulation process 158), and in the master control process. This TCL code interacts with the slave processes and the user interface to coordinate all slaves into a coherent simulation verification system. Each slave receives commands from the master control process via the slave's standard input. The slave communicates to the master control process by writing properly formatted text strings to the slave's standard output, or, alternatively, by writing large amounts of data (for example, B.D.D.s generated by the symbolic simulator module) into data files formatted for a specific task.

The random simulation module, sometimes referred to as the "VCS" module, comprises the DUT model, the design environment model (together the design environment model and the DUT model make up the model under test, or the "MUT"), and an upper level control program called CoverBooster. The DUT model, the environment model, and the interface object between them are each specific to the design being verified; the CoverBooster control program is design independent. The DUT model is described in the Verilog HDL and is simulated on a Synopsys product called VCS. The environment model, which serves to generate valid random input vectors for the DUT, is modeled by the user in the VERA Hardware Verification Language (the "Vera language"), and comprises a reset function and a drive function.

A comprehensive explanation of the Verilog HDL language and its capabilities is presented in "IEEE Standard Hardware Description Language Based on the Verilog Hardware Description Language," IEEE Standard 1364-1995, Institute of Electrical and Electronic Engineers, Oct. 1996, which is herein incorporated by reference.

The Vera language is a product of Synopsys, Inc., of Mountain View, California. A comprehensive explanation of the Vera language and its capabilities is presented in the Vera User's Manual incorporated herein as Appendix 2.

The CoverBooster program, a Synopsys product written in the Vera language, serves to control the random simulation process by linking with and controlling the DUT, and interfacing the VCS slave process with the rest of the simulation verification system. Under the control of CoverBooster, the drive function in the model environment generates a random input vector which is a valid input for the DUT model. During simulation, the DUT model is driven by these input vectors into different states in a search for goal states. In particular, CoverBooster drives the DUT model into the reset state, as directed by master control process 154; records input vectors applied to the DUT model; reads traces from the formal simulation modules in order to drive the DUT model into desired states; advances the DUT model clock; writes out completed trace sets for later use; and replays previously-generated and saved trace sets.

The symbolic simulation module, the satisfiability module, and the unreachability module 160, 162, and 164 are written in C, C++, pascal, fortran, or any other language suitable for accomplishing the purposes described herein. For convenience, they are preferably written in the same language of as many of the processes which control them as possible, particularly the hybrid verification program used to interpret TCL control procedures.

To generate a sequence of input vectors suitable for driving the DUT into each of a defined set of goal states, a user first defines a DUT model, a model environment, and a set of design goal states. The user then integrates the DUT and the model environment into a random simulation module. The user also defines the heuristics to be used by the master control process in guiding the verification process. For example, the user can designate the following heuristics:

> start the simulation process, beginning with a number of cycles of random searching from the reset state, in order to identify as many goal states as possible as quickly as possible, and thereafter
>
> proceed to a satisfiability analysis from the reset state, the SAT analysis being allowed to proceed until all goal states are reached, and input vectors

identified, or until CPU time or memory resource limits are exceeded;

thereafter symbolic simulation is run from the reset state and allowed to proceed until all goal states are reached and input vectors identified or until CPU time or memory resource limits are exceeded; and

thereafter random simulation is resumed from the reset state, followed by formal simulation from the goal state most recently identified by random simulation, or the last state reached by random simulation, and a process of interleaving between random simulation, satisfiability techniques, and symbolic simulation is begun, and continued until input sequences for all goal states are identified.

It is generally preferred that heuristics be set such that demands on memory and CPU time are progressively increased as the simulation verification process continues. This has been found, in general, to result in the most rapid identification of goal state inputs possible in the initial stage of the analysis process, and to leave the more exhaustive stages, in which the rate at which goal states are reached tends to drop, to later.

Optionally, the user may specify, via the user interface, that an unreachability analysis be initiated and conducted simulataneously with the random and formal simulation processes.

Heuristics used to guide the simulation process are largely implemented in

TCL code running in the master control process. This TCL code allows for many user-specified configuration settings, sometimes known as TCL "knobs", and tracks session progress and status by monitoring TCL "variables" which change as the session progresses. For example, the number of times random simulation is to be started from the reset state in the heuristic scenario described above is specified by the user via the user interface, by setting a variable "initRunsMax." Similarly, the number of simulation cycles to be allowed without reaching a goal state during any of the starts from reset is specified in variable 'initCycles'. The variable 'initCycles' specifies that random simulation module 158 stop when a given number of DUT clock cycles have been simulated without hitting a previously unreached goal. It has been determined that a suitable value for initRunsMax, for most reasonably complex DUT models, is 20. Similarly, a beneficial number of times random simulation is to be allowed to execute (that is, to drive the DUT model) without striking a goal state has been found to be about 10,000.

When the user wishes to conduct unreachability analyses, the TCL unreach knob is set to 'unreach_lmbm'. This causes the master control process to initiate unreachability analysis, preferably simultaneously with the beginning of formal simulation.

All TCL knobs and variables are organized as elements of a single global array variable named "ctgInfo." Preferably, when a TCL knob or global variable is added to the ctgInfo array, a documentation entry is also added to aid later designers or analysts in evaluating the simulation process used.

To begin analysis the user initializes the system by invoking the VCS slave module 158 via the master control process and the user interface, through use of the UNIX 'fork' command. The "hv-I-master" command is issued to open a pipe to read/write the standard input and output of the master control process and to define hv processes as master, not slave, processes. The TCL file "hvRecipe.tcl" is loaded, to define procedures needed to implement the various high-level master control methods; the master control process is told the values of user-specified session parameters, including heuristics used to guide the simulation process and the set of defined goal states; and by calling the "hvino_create_hnl_design" command the DUT model is loaded. The formal methods slave processes to be started at the completion of random simulation are selected by sending "hv_setup_formal" to the master control program. The selected slave processes are started, loaded with HNL and the set of defined goals, and configured as necessary.

The verification process is initiated with random simulation via the user interface, through use of the 'hv_start_vcs_session' TCL procedure. Goals are loaded into the VCS process by use of the PLI routine "$defineFsmHV" and the DUT is driven to the reset state. The master control process defines simulation coverage progress metrics and initiates random simulation from the reset state. The random simulation module initiates monitoring of the reaching of goal states through the use of PLI functions. The reaching of goal states is generally monitored by designating selected registers within the DUT model, comprising signals in the goal set, as having callback nodes. The changing of the state (i.e., the value) of any of these callback nodes causes a check to be made to determine whether a goal state has been reached. Preferably this check is made at a consistent point, at least once per cycle, in the DUT model time cycle, for example the "T/4" point in the DUT model time cycle shown in Figure 6. If so, the random simulation module 158 is instructed to write the input vector which reached the goal state into the trace file. (Again, preferably all states generated in the DUT model are recorded in the trace set, to facilitate later re-recreations of goal states.) Thus the relatively inefficient method of continuously checking to determine whether a goal state has been reached is avoided. Instead, it is only when one of the goal set register variables has changed that the DUT model state is checked to determine whether a goal state has been reached.

A component of random simulation module 158 used in controlling the random simulation process 158 is the CoverBooster program. A portion of top-level control flow within the CoverBooster program is shown in pseudocode in Figure 5. CoverBooster is configured to drive the random simulator under a variety of conditions, one or more of which typically arise during any given verification session, depending upon heuristics defined by the user and received from the master control process.

At 301 CoverBooster program 300 initializes the DUT model and reads program invocation arguments received from the master control program which determine the exact function to be performed by CoverBooster at this particular call. At 302 two DUT model clock cycles are allowed to pass prior to beginning simulation, in order to ensure that other, parallel processes have completed and that the system is ready.

One option enabled by CoverBooster is the driving of the DUT model into or through an entire set of predetermined states, as for example, a previously-defined set of goal states, given a suitable input vector. This ability is used, for example, in verifying the DUT after input vectors for each of the goal states has been identified. At 303 CoverBooster determines whether such a test case replay has been requested. If so, the trace vector is replayed at 304 and at 305 control is returned to the master control process.

Loop 306 provides control mechanisms for all actions needed for efficient coverage of goal states. The loop begins with statement 307. At 308 a read of control flags is performed. These control flags, set and monitored by the master control program, are used by the master control program for guiding the random analysis.

For example, if the random simulation module is only being requested to drive the DUT model to the reset state, the control flag read at 308 causes at 309 a call to the reset function and at 310 a resetting of the ResetFlag to inform the master control program that the reset has been completed.

Optional biasing considerations set by the user and/or by the environment model are considered at 311, and appropriate action is taken. Biasing of input may be accomplished in at least two ways. The first is through the use of the "UserBiasDrive" function. A totally random input for any particular bit of an input vector would have a value of "1" in 50% of cases and a value of "0" in the remainder. The user may change this ratio, or bias the input, by using standard Vera language techniques within the "UserBiasDrive" function, as described in the Vera language manual of Appendix 2. In addition, CoverBooster provides a "dynamic" biasing mechanism. When a trace for driving the DUT model into goal state is provided from a formal engine, the "biasWeightsFlag" is set in the random simulator, prior to driving the DUT model through the trace. This causes CoverBooster to count, for each input signal, how many times the signal "1" or "0" is encountered as the trace is applied. These counts, or "weights", may then be used, through application of other standard Vera language techniques, to bias the inputs so that something other than a 50%-50% ratio of "1"s and "0"s results. These "dynamically biased" inputs are used in the first few (e.g., 1000) input patterns generated following the reaching of a goal state. Periodically these weights are cleared, as at 311, since their effectiveness decreases as the number of test cycles increases.

If analysis is returning to random simulation following location of a goal state by a formal engine, at 312 the trace provided by the formal engine is replayed to provide a resumption point for random analysis.

Quit and stop flags set for various control and heuristic reasons are considered at

313. For example, if the master process determines that simulation should be paused, or if PLI processes determine that a simulation progress coverage metric is no longer being satisfied, the "stopFlag" is set. At 313 CoverBooster checks the flag, and if the flag is set breaks out of loop 306 so that further direction or input from the master process can be awaited and appropriate action taken. Similarly, if the master process determines, as for example by interactive user request, that the test generation session is to be terminated, the "quitFlag" is set, and upon determining at 313 that the flag is set CoverBooster stops execution of loop 306 and the session is terminated.

At 314, if no stop or quit flags have been encountered, a biased, constrained random input sequence is generated and the DUT model is driven through a single clock cycle.

At 315 a choice is made between default user-specified biasing options, which are applied for the first 1000 (or other suitable number) of clock cycles following reaching of a goal state, or dynamic, automatically-generated biases, and the appropriate option is used for generating the input vector and driving the DUT model, either at 316 or 317 as appropriate. The drive message is passed to the DUT interface object, with appropriate parameters; the interface object uses the parameters to control mapping of a legal input vector for the DUT model. The legal input vector is applied to the DUT model, and the resultant state of the DUT model is determined. If one of the callback nodes has changed value, a determination is made as to whether the changing of the register state signifies the reaching of a goal state. If a goal state has been reached, at 318 the fact that a state has been reached is reported to the master control process.

At 319 – 320 the clock is updated and timing is judged for the appropriate point in the clock cycle for starting the next simulation cycle. Timing of the application of inputs to the DUT model, as specified at lines 320, is shown in Figure 6. Solid line 410 represents the pulsing of the clock circuit. A clock cycle begins at positive edge 401 of the clock pulse. At halfway point 402 of the cycle the clock pulse undergoes a negative edge. At point 403 the next clock cycle begins with positive edge 401'. To ensure suitable reading and proper driving, inputs are applied at 405, ¾ of the way through the cycle, and the DUT state is checked at point 404, ¼ of the way into the clock cycle.

Loop 306 of the CoverBooster program, as shown in Figure 5, is executed until it is determined that all goal states have been reached or have been proved unreachable or the simulation coverage progress metric is no longer satisfied.

At the completion of the random simulation cycle control is returned to the master process. What happens next is specified by heuristics previously set by the user or by the master control process, in light of the state reached by the verification simulation process.

After random simulation has been started from the reset state initRunsMax times (that is, 20 times in this Example), the master control process proceeds to MODE_INIT_SAT. To this end the user interface has previously set the TCL knob "satCycMax"' to a value > 0. MODE_INIT_SAT phase uses the SAT engine to search for goals, starting from the reset state. The resource limits "satCycMax" and "satCpuMax," previously set by the user interface or by master control process 154, are used as coverage progress metrics for stopping analysis. If a goal is reached prior to exhaustion of CPU or cycle resources, the trace is written to the trace vector and is played by the VCS module, as previously explained. Random simulation then proceeds until 'initCycles' system clock cycles have passed without hitting a new goal. Then VCS resets and waits for another formal search to be carried out. If SAT fails to reach a goal from the reset state, master control process 154 proceeds to the next phase, MODE_INIT_SYM.

Operation of the MODE_INIT_SYM phase is similar to MODE_INIT_SAT, except that the symbolic simulation search engine is used, with resource limits being defined by the 'symsimCpuMax' and 'symsimCycMax' TCL variables. If traces to no new goal states are identified, master control process 154 causes the DUT model to be driven again to the reset state and proceeds to the next phase, MODE_SAT.

In MODE_SAT, VCS simulates randomly for a determined number of cycles and stops; the last state reached is written to a file and is used as a starting point for satisfiability analysis. The satisfiability module 162 begins from this state. If the satisfiability module identifies a new goal state prior to exceeding memory or CPU limits, the trace is replayed by VCS and random simulation is resumed. If no new goal state is reached, master control process 154 proceeds to MODE_SYMSIM.

MODE_SYMSIM is similar to MODE_SAT, except that symbolic simulation is substituted for use of satisfiability techniques. Again, VCS simulates for a determined number of cycles and stops, starting from the last identified goal state; the last state reached by random simulation is then written to a file and is used as a starting point for symbolic simulation. Symbolic simulation module 160 is started; if a new goal state is reached prior to exhaustion of CPU or memory resources, the trace is replayed VCS and random simulation is resumed. Otherwise MODE_SYMSIM_SAT is initiated.

MODE_SYMSIM_SAT is similar to MODE_SAT and MODE_SYMSIM, except that after starting from the last state reached by random simulation, the verification process alternates between symbolic simulation and satisfiability techniques. When both symbolic simulation and satisfiability techniques have interleaved a given consecutive number of times (stored in variable 'formalMissThresh') without identifying a new goal state, the DUT model is driven into either the reset state or the last state reached by random simulation, MODE_SYMSIM is entered, and the process is repeated until all goal states have been identified or proved unreacheable.

When all goal states have either been reached or proved unreachable, MODE_DONE is set by the master control process. The PLI command $reportDutStateHV is called by master control process 154 to report the current DUT state to the trace vector file and any remaining unreported reached goals are reported using the PLI command $reportNewCoverDataHV. Control is returned to the user for further action, if desired.

It must be borne in mind that the heuristic framework presented in the Example is only an example, and that many heuristic approaches to the simulation verification process are possible and desirable, depending upon the particulars of the DUT and the test conditions. Figure 7 is a pseudocode representation of a general process for controlling interleaving between analysis methods. In particular, Figure 7 illustrates the interleaving process conducted in TCL by master control process 154 when one of the formal engines stops and random simulation is to be run, regardless of whether the stop is due to a new goal state being reached or whether one of the formal search coverage progress metrics has stopped being satisfied. This pseudocode is written so as to incorporate use of the Synopsys VCS Verilog simulator.

At 330, 340, and 350 it is determined which mode the simulation verification process has stopped in. If the formal verification simulation process has stopped in MODE_INIT_SYM, the determination is made at 331, 335 as to whether a goal state has been reached or simulation has stopped due to non-satisfaction of a coverage metric. If no new goal state has been identified, at 332 a determination is made, based on the previously-decided heuristics, which formal method is to be used after the next phase of random simulation that is about to be started has completed. At 333 the next phase of random simulation is started. Upon the termination of this random simulation phase due to non-satisfaction of coverage progress metrics, the next formal method chosen at 332 is initiated. If a new goal state has been identified, at 336 the trace which lead to identification of the new goal state is used to drive the DUT model into that goal state and at 338 random simulation selected is resumed.

If the formal verification simulation process has stopped in MODE_INIT_SAT, a similar process is used. The determination is made at 341, 345 as to whether a goal state has been reached or a formal method has stopped due to non-satisfaction of a coverage progress metric. If no new goal state has been identified, at 342 a determination is made, based on the previously-decided heuristics, which formal method is to be used after the next phase of random simulation that is about to be started has completed, and at 343 random simulation is resumed. Upon termination of random simulation due to non-satisfaction of coverage progress metrics, the chosen formal method is started. If a new goal state has been identified, at 346 the trace which lead to identification of the new goal state is used to drive the DUT model into that goal state and at 348 random simulation is resumed.

If the formal verification simulation process has stopped in MODE_SAT, MODE_SYMSIM_SAT, or MODE_SYMSIM, and the previous formal search method has identified no goal states, at 352, 353 a determination is made as to whether the last formal process was MODE_SAT or MODE_SYMSYM_SAT. If MODE_SAT, MODE_SYMSIM_SAT is designated as the next phase. If MODE_SYMSIM_SAT, MODE_SYMSIM is designated as the next phase. In either case at 355 a determination is made as to whether the predetermined number of interleaving cycles "formalMissThresh" has been reached; if so, at 356 the random simulation module is called to drive the DUT model to the reset state and at 365 random simulation is resumed. If 'formalMissThresh' has not been reached and the last formal engine used was symbolic simulation, at 358 VCS is used to replay the latest input trace and at 365 formal simulation is resumed from the latest identified goal state, using the formal method designated at 352, 353.

If the last formal search process used has found a goal state and 'formalHitThresh' has been reached, at 364, 365 bias weights are cleared after trace replays and VCS is called to resume random simulation from the last state reached.

While the invention has been described and illustrated in connection with preferred embodiments, many variations and modifications as will be evident to those skilled in this art may be made without departing from the spirit and scope of the invention, and the invention is thus not to be limited to the precise details of methodology or construction set forth above as such variations and modification are intended to be included within the scope of the invention.